Universidad Nacional de Colombia - sede Bogotá
Facultad de Ingeniería
Departamento de Sistemas e Industrial
Curso: Ingeniería de Software 1 (2016701)

Testing Linter TheBigBro:

●Tomás Nicolás Jerez García.
●Jacobo Ulises Cortés Duque.
●Santiago Zamora Garzón.
●Gabriel Mateo Gonzalez Lara.

Herramienta Utilizada

La herramienta empleada para la validación del estilo y la calidad del código fue Flake8, un linter ampliamente utilizado en proyectos Python debido a su integración con estándares PEP8, su extensibilidad y su capacidad para detectar errores sintácticos, problemas de estilo y patrones perjudiciales para el mantenimiento del código. Flake8 combina la funcionalidad de PyFlakes, mccabe y pycodestyle, lo que permite obtener una revisión completa del código fuente.

Configuración Aplicada

La verificación se ejecutó utilizando la configuración definida en el archivo `setup.cfg`. La configuración establece parámetros esenciales para adaptar el linter a las necesidades del proyecto. Entre las reglas personalizadas y ajustes aplicados se encuentran:

Parámetros Configurados

- **`max-line-length = 130`**: Se define un límite de 130 caracteres por línea, permitiendo mayor flexibilidad en declaraciones extensas sin comprometer la legibilidad.
- **`exclude = .git, __pycache__, .venv, Proyecto/.venv, Proyecto/tests/__pycache__, */migrations/`**: Se excluyen directorios que no forman parte del código fuente relevante, como carpetas de entorno virtual, cachés, migraciones y control de versiones.
- **`extend-ignore = E203, W503`**: Se ignoran dos reglas específicas:
  - **E203**: Relacionada con espacios antes de los slices; se omite porque está en conflicto con el formato generado por Black.
  - **W503**: Advierte sobre saltos de línea antes de operadores; se deshabilita ya que se prefiere el estilo recomendado por PEP8.

Justificación de las Reglas Personalizadas

Las dos excepciones incluidas se consideran estándar en proyectos que utilizan Black y buscan mantener consistencia entre ambos formateadores. No se añadieron

configuraciones adicionales ni reglas más restrictivas, lo cual mantiene el proceso simple pero alineado con el estilo moderno de Python.

Resultados Obtenidos

Durante la ejecución inicial del linter se detectó un conjunto significativo de errores relacionados principalmente con formato, espacios en blanco y organización del código. Entre los hallazgos más relevantes identificados por Flake8 se encuentran:

- **E302 / E301 / E305**: Problemas con la cantidad de líneas en blanco antes o después de definiciones de funciones o clases.
- **W293**: Líneas en blanco que contienen espacios o tabulaciones.
- **W291**: "Trailing whitespace", es decir, espacios innecesarios al final de la línea.
- **W292**: Falta de una nueva línea al final del archivo.
- **F841**: Variables asignadas pero nunca utilizadas.

Las capturas proporcionadas muestran estos errores repetidos principalmente en los archivos:

- `core/models.py`
- `core/views.py`
- `tests/test_business_rules_contacts.py`

Lo anterior evidencia que el proyecto presentaba inconsistencias comunes en espacios, saltos de línea y limpieza de variables, lo cual es típico antes de aplicar una revisión formal del linter. Durante la ejecución inicial del linter, se identificaron diversos errores de estilo, advertencias y problemas menores relacionados principalmente con:

- Longitudes de línea que excedían el máximo permitido.
- Espacios innecesarios o ausentes en expresiones y operadores.
- Importaciones no utilizadas.
- Convenciones de nombres no alineados con PEP8.
- Saltos de línea inconsistentes.

Estos hallazgos fueron corregidos manualmente o mediante herramientas automáticas, según correspondía. No quedaron errores sin resolver.

Evidencia de Ejecución

Se cuenta con capturas de pantalla del resultado arrojado por Flake8 durante la fase de análisis, donde se muestran los errores detectados antes de las correcciones. Dichos

screenshots evidencian que el linter fue ejecutado sobre el proyecto completo respetando las reglas configuradas.

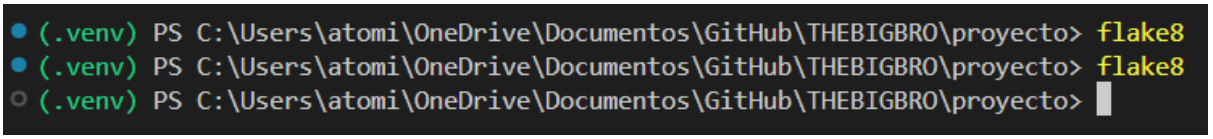
```
(.venv) PS C:\Users\atomi\OneDrive\Documentos\GitHub\THEBIGBRO\proyecto> flake8
.\core\models.py:4:1: E302 expected 2 blank lines, found 1
.\core\models.py:17:1: E302 expected 2 blank lines, found 1
.\core\models.py:22:1: E302 expected 2 blank lines, found 1
.\core\models.py:35:1: E302 expected 2 blank lines, found 1
.\core\models.py:37:5: E301 expected 1 blank line, found 0
.\core\models.py:39:5: E301 expected 1 blank line, found 0
.\core\models.py:42:1: E302 expected 2 blank lines, found 1
.\core\models.py:47:5: E301 expected 1 blank line, found 0
.\core\models.py:49:5: E301 expected 1 blank line, found 0
.\core\models.py:52:1: E302 expected 2 blank lines, found 1
.\core\models.py:62:5: E301 expected 1 blank line, found 0
.\core\models.py:64:5: E301 expected 1 blank line, found 0
.\core\models.py:67:1: E302 expected 2 blank lines, found 1
.\core\models.py:72:5: E301 expected 1 blank line, found 0
.\core\models.py:74:5: E301 expected 1 blank line, found 0
.\core\models.py:77:1: E302 expected 2 blank lines, found 1
.\core\models.py:83:5: E301 expected 1 blank line, found 0
.\core\models.py:86:5: E301 expected 1 blank line, found 0
.\core\models.py:95:1: E302 expected 2 blank lines, found 1
.\core\models.py:102:5: E301 expected 1 blank line, found 0
.\core\models.py:105:1: E302 expected 2 blank lines, found 1
.\core\models.py:125:5: E301 expected 1 blank line, found 0
.\core\models.py:128:1: E302 expected 2 blank lines, found 1
.\core\models.py:134:5: E301 expected 1 blank line, found 0
.\core\models.py:148:1: E302 expected 2 blank lines, found 1
.\core\models.py:152:5: E301 expected 1 blank line, found 0
.\core\views.py:102:1: E302 expected 2 blank lines, found 1
.\core\views.py:300:1: W293 blank line contains whitespace
.\core\views.py:303:1: W293 blank line contains whitespace
.\core\views.py:320:1: W293 blank line contains whitespace
.\core\views.py:459:13: F841 local variable 'tolerance' is assigned to but never used
.\core\views.py:473:1: W293 blank line contains whitespace
```

```
.\core\views.py:476:1: W293 blank line contains whitespace
.\core\views.py:481:1: W293 blank line contains whitespace
.\core\views.py:528:1: W293 blank line contains whitespace
.\core\views.py:550:1: W293 blank line contains whitespace
.\core\views.py:576:1: W293 blank line contains whitespace
.\core\views.py:580:1: W293 blank line contains whitespace
.\core\views.py:1562:1: W293 blank line contains whitespace
.\core\views.py:1572:1: W293 blank line contains whitespace
.\core\views.py:1582:21: F841 local variable 'user' is assigned to but never used
.\core\views.py:1595:1: W293 blank line contains whitespace
.\core\views.py:2351:1: E302 expected 2 blank lines, found 1
.\core\views.py:2357:1: W293 blank line contains whitespace
.\core\views.py:2387:1: W293 blank line contains whitespace
.\core\views.py:2403:44: W292 no newline at end of file
```

```
.\tests\test_business_rules_contacts.py:6:23: W291 trailing whitespace
.\tests\test_business_rules_contacts.py:7:20: W291 trailing whitespace
.\tests\test_business_rules_contacts.py:12:27: W291 trailing whitespace
.\tests\test_business_rules_contacts.py:13:29: W291 trailing whitespace
.\tests\test_business_rules_contacts.py:14:34: W291 trailing whitespace
.\tests\test_business_rules_contacts.py:15:37: W291 trailing whitespace
.\tests\test_business_rules_contacts.py:44:1: E302 expected 2 blank lines, found 1
.\tests\test_business_rules_contacts.py:66:1: E302 expected 2 blank lines, found 1
.\tests\test_business_rules_contacts.py:81:53: W292 no newline at end of file
```

5. Evidencia de Corrección

Posteriormente, se ejecutó nuevamente Flake8, obteniéndose un reporte limpio, sin errores ni advertencias. Este resultado validó que todas las inconsistencias encontradas previamente fueron solucionadas.



6. Conclusión

La implementación de Flake8 permitió validar la calidad del código y asegurar el cumplimiento de estándares de estilo en el proyecto. Gracias a la configuración aplicada y a la corrección de todos los hallazgos, el repositorio mantiene un formato consistente, legible y alineado con las mejores prácticas de Python. Esta verificación contribuye a mejorar el mantenimiento futuro del proyecto y reduce la probabilidad de errores derivados de inconsistencias en el estilo de codificación.